#include<iostream>

#include<string>

#include<windows.h>

using namespace std;

class node{

    public:

    int shore;

    int thieves[3];

    int bags[3];

    int money[3];

    int visited;

    string steps;

    node \*next;

    node(){

        visited=0;

        shore=0;

        steps="";

        for(int i=0;i<3;i++){

            thieves[i]=0;

            bags[i]=0;

        }

        money[0]=1000,money[1]=700,money[2]=300;

    }

    node(node \*p){

        this->visited = p->visited+1;

        this->shore=p->shore;

        this->steps=p->steps;

        for(int i=0;i<3;i++){

            this->thieves[i]=p->thieves[i];

            this->bags[i]=p->bags[i];

        }

        this->money[0]=1000,this->money[1]=700,this->money[2]=300;

    }

    int legal(){

        int tot,f;

        for(int i=0;i<2;i++){

            tot=0,f=0;

            for(int j=0;j<3;j++){

                if(thieves[j] == i){

                    tot+=money[j];

                    f=1;

                }

                if(bags[j] == i)

                    tot-=money[j];

            }

            if(tot < 0 && f==1)

                return 0;

        }

        return 1;

    }

    int end(){

        int f=0;

        for(int i=0;i<3;i++){

            if(thieves[i]==1)

                f++;

            if(bags[i]==1)

                f++;

        }

        return f==6;

    }

    void disp(){

        cout << "t: ";

        for(int i=0;i<3;i++)

            cout << thieves[i] << ' ';

        cout << endl;

        cout << "b: ";

        for(int i=0;i<3;i++)

            cout << bags[i] << ' ';

        cout << endl;

    }

};

class stack{

    public:

        int size;

        node \*head;

        stack(){

            size=0;

            head=NULL;

        }

        void push(node \*pnn){

            size++;

            pnn->next=head;

            head=pnn;

        }

        void backup(node \*pnn){

            node \* cp = new node(pnn);

            this->push(cp);

        }

        node \*pop(){

            size--;

            node \*pnn = head;

            head = pnn->next;

            return pnn;

        }

        int found(node\* search){

            node\* trav = head;

            while(trav != NULL){

                if(search->shore == trav->shore){

                    if(trav->thieves[0]==search->thieves[0] && trav->thieves[1]==search->thieves[1] && trav->thieves[2]==search->thieves[2]

                    && trav->bags[0]==search->bags[0] && trav->bags[1]==search->bags[1] && trav->bags[2]==search->bags[2]){

                        return 1;

                    }

                }

                trav = trav->next;

            }

            return 0;

        }

        void disp(){

            node \*t = head;

            while(t != NULL){

                t->disp();

                cout << endl;

                t = t->next;

            }

        }

};

void expand(node \*curr,stack \*S,stack \*Sol,stack \*mem){

    int to;

    node \*child;

    for(int i=0;i<3;i++){

        if(curr->thieves[i] == curr->shore){

            //with each bag

            for(int b=0;b<3;b++){

                if(curr->bags[b] == curr->shore){

                    child = new node(curr);

                    if(child->shore==0)

                        child->shore=1;

                    else

                        child->shore=0;

                    child->thieves[i]=child->shore,child->bags[b]=child->shore;

                    child->steps += to\_string(child->visited)+".thieve "+to\_string(i)+" bag "+to\_string(b)+"-->"+to\_string(child->shore)+" ";

                    if(child->legal()){

                        if(child->end()){

                            Sol->push(child);

                        }else{

                            if(!mem->found(child)){

                                mem->backup(child);

                                S->push(child);

                            }else{

                                delete child;

                            }

                        }

                    }else{

                        delete child;

                    }

                }

            }

            //with thieves

            for(int t=0;t<3;t++){

                if(curr->thieves[t] == curr->shore && t!=i){

                    child = new node(curr);

                    if(child->shore==0)

                        child->shore=1;

                    else

                        child->shore=0;

                    child->thieves[i]=child->shore,child->thieves[t]=child->shore;

                    child->steps += to\_string(child->visited)+".thieve "+to\_string(i)+" thieve "+to\_string(t)+"-->"+to\_string(child->shore)+" ";

                    if(child->legal()){

                        if(child->end()){

                            Sol->push(child);

                        }else{

                            if(!mem->found(child)){

                                mem->backup(child);

                                S->push(child);

                            }else{

                                delete child;

                            }

                        }

                    }else{

                        delete child;

                    }

                }

            }

            //thieve back off

            child = new node(curr);

            if(child->shore==0)

                child->shore=1;

            else

                child->shore=0;

            child->thieves[i]=child->shore;

            child->steps += to\_string(child->visited)+".thieve "+to\_string(i)+"-->"+to\_string(child->shore)+" ";

            if(child->legal()){

                if(child->end()){

                    Sol->push(child);

                }else{

                    if(!mem->found(child)){

                        mem->backup(child);

                        S->push(child);

                    }else{

                        delete child;

                    }

                }

            }else{

                delete child;

            }

        }

    }

}

void solve(stack \*S,stack \*Sol,stack \*mem){

    node \*pnn = new node,\*curr;

    S->push(pnn);

    mem->backup(pnn);

    while(S->head != NULL){

        curr = S->pop();

        expand(curr,S,Sol,mem);

    }

}

int main(){

    stack \*S = new stack,\*Sol = new stack,\*mem = new stack;

    solve(S,Sol,mem);

    node \*t = Sol->head;

    cout << "------solutions----------" << endl;

    while(t != NULL){

        cout << t->steps << endl << endl;

        t=t->next;

    }

    cout << "------solutions----------" << endl;

    cout << Sol->size << " solutions " << endl;

    delete S;

    delete Sol;

    delete mem;

    return 0;

}